

Python Programming Tools (OOP)

Fortgeschrittene Programmierung mit Python 3

Autor: Stefan Berger
(Ergänzt durch Walter Rothlin)

HWZ

Die Hochschule für Wirtschaft
in Zürich

Einführung in OOP mit Python

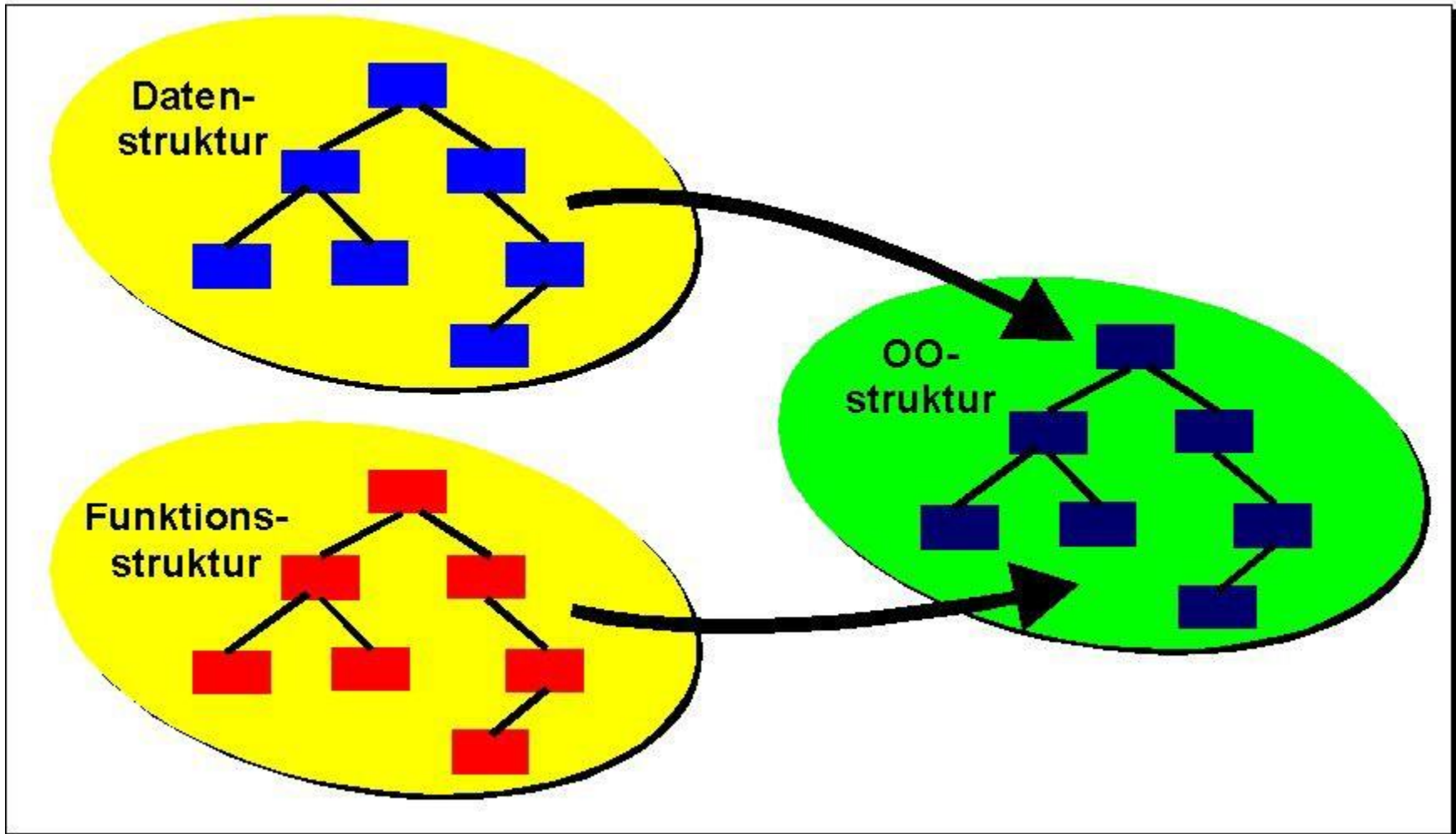


Was	Kurze Einführung in OOP
Lernziel	Begriffe der OOP kennen
Warum	Python ermöglicht auch objektorientiertes Programmieren
Ablauf	Präsentation Dozent

OOP - Inhalt

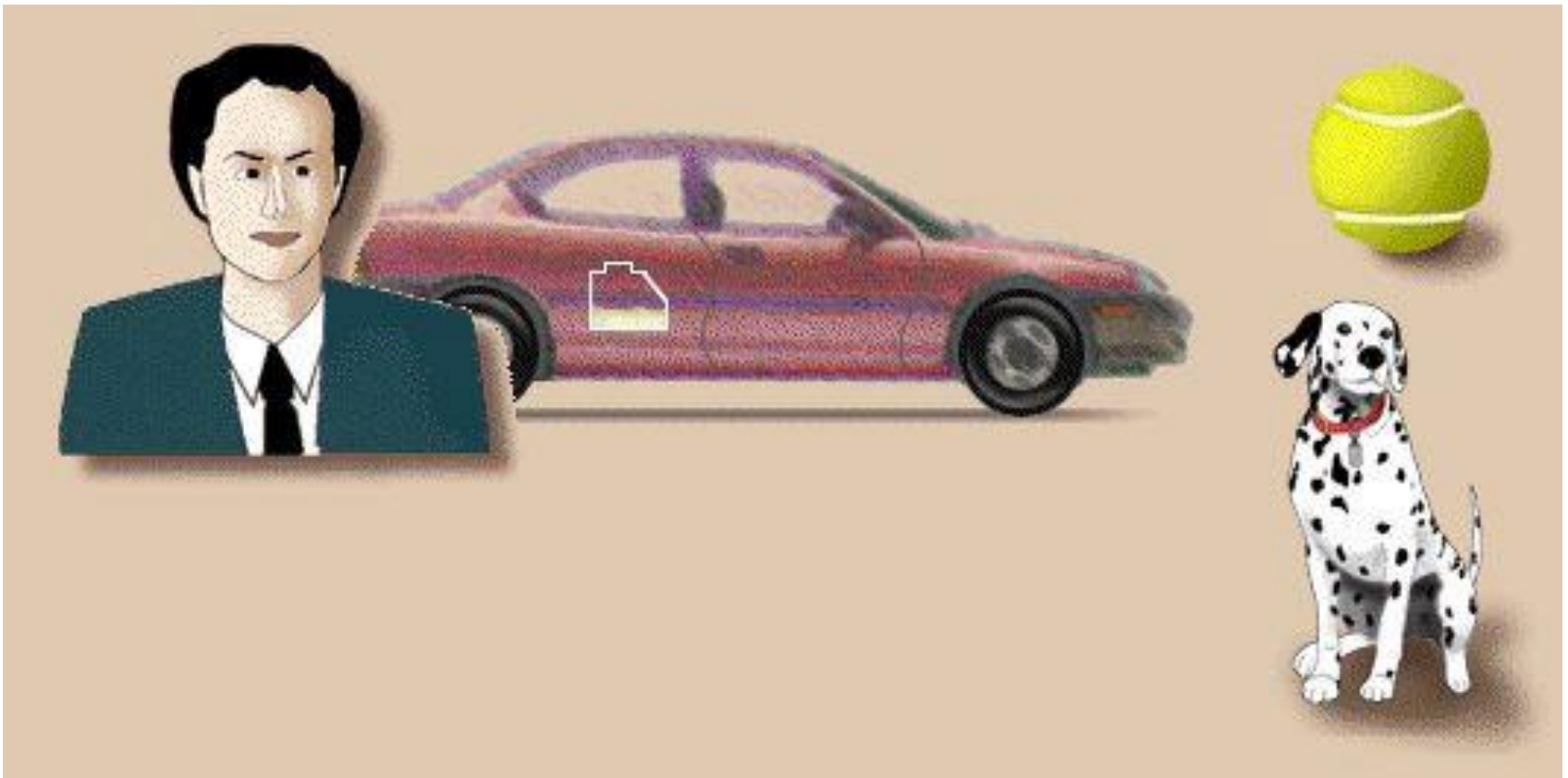
- ❑ Was ist anders?
- ❑ Was sind Objekte?
- ❑ Was sind Methoden?
- ❑ Was versteht man unter Nachrichten?
- ❑ Was ist eine Klasse?
- ❑ Wie findet man Objekte?

OOP - was ist anders ?



OOP - was sind Objekte ?

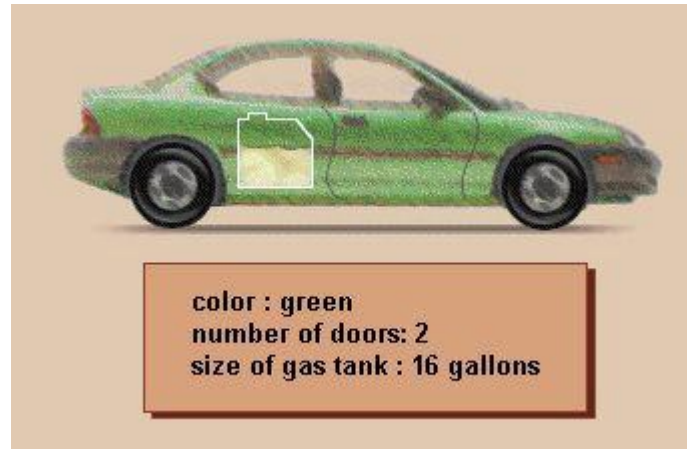
- ❑ Gegenstände, Dinge der realen Welt
- ❑ aber auch Verfahren, Algorithmen, virtuelle Konzepte



OOP - was sind Objekte (2) ?

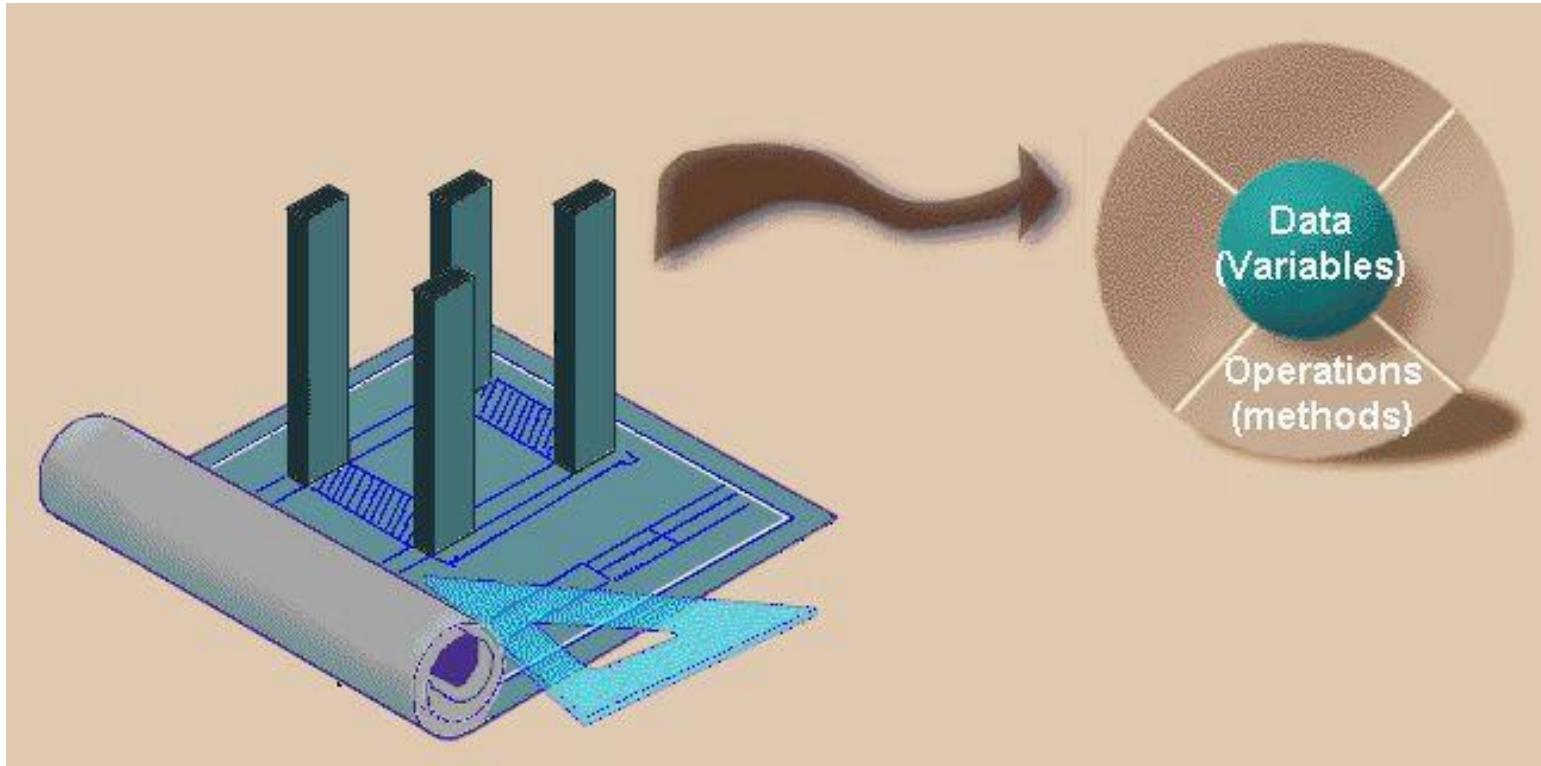
- ❑ Objekt:
 - ❑ besitzt Eigenschaften und Zustände (Attribute)
 - ❑ kann durch Operationen (Methoden) manipuliert werden oder Informationen preisgeben

- ❑ Das Objekt selbst ist zuständig für die Speicherung und Verwaltung seiner Attribute (Daten - nach Prinzip: teile und herrsche)



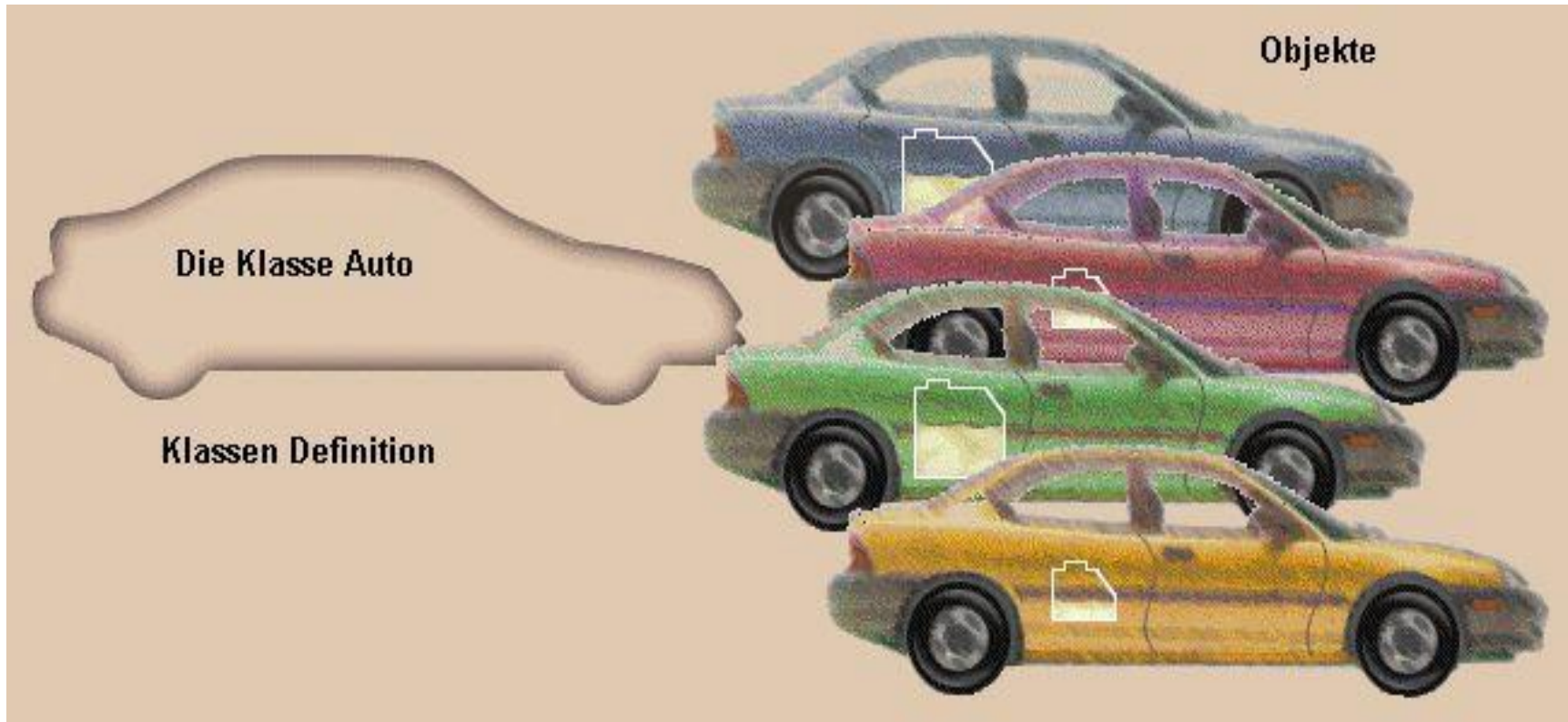
OOP - Was ist eine Klasse ?

- ❑ Klasse ist eine Vorlage, Template, Schablone oder Bauplan
- ❑ Klasse definiert Attribute und Methoden



OOP - Was ist eine Klasse (2) ?

- ❑ Aus Klassen werden die Objekte instanziiert (Instanziierung)
- ❑ Jedes Objekt verfügt über eine eigene Identität



OOP - Wie findet man Klassen ?

- ❑ Das Finden von Klassen / Objekten ist schwierig
- ❑ Iterativer Prozess
- ❑ Resultat wird in Objektmodell festgehalten
- ❑ UML (Unified Modeling Language)
- ❑ Einfachster Ansatz
(Nomen / Substantiv -> Objekt, Verben -> Methoden)
- ❑ Design Patterns (Entwurfsmuster)

OOP - Beispiel Klasse: Switch

```
class Switch:
    def __init__(self):
        self.state = False

    def on(self):
        self.state = True

    def off(self):
        self.state = False

    def is_on(self):
        return self.state
```

OOP - Fragen

☐ Was ist eine Klasse?

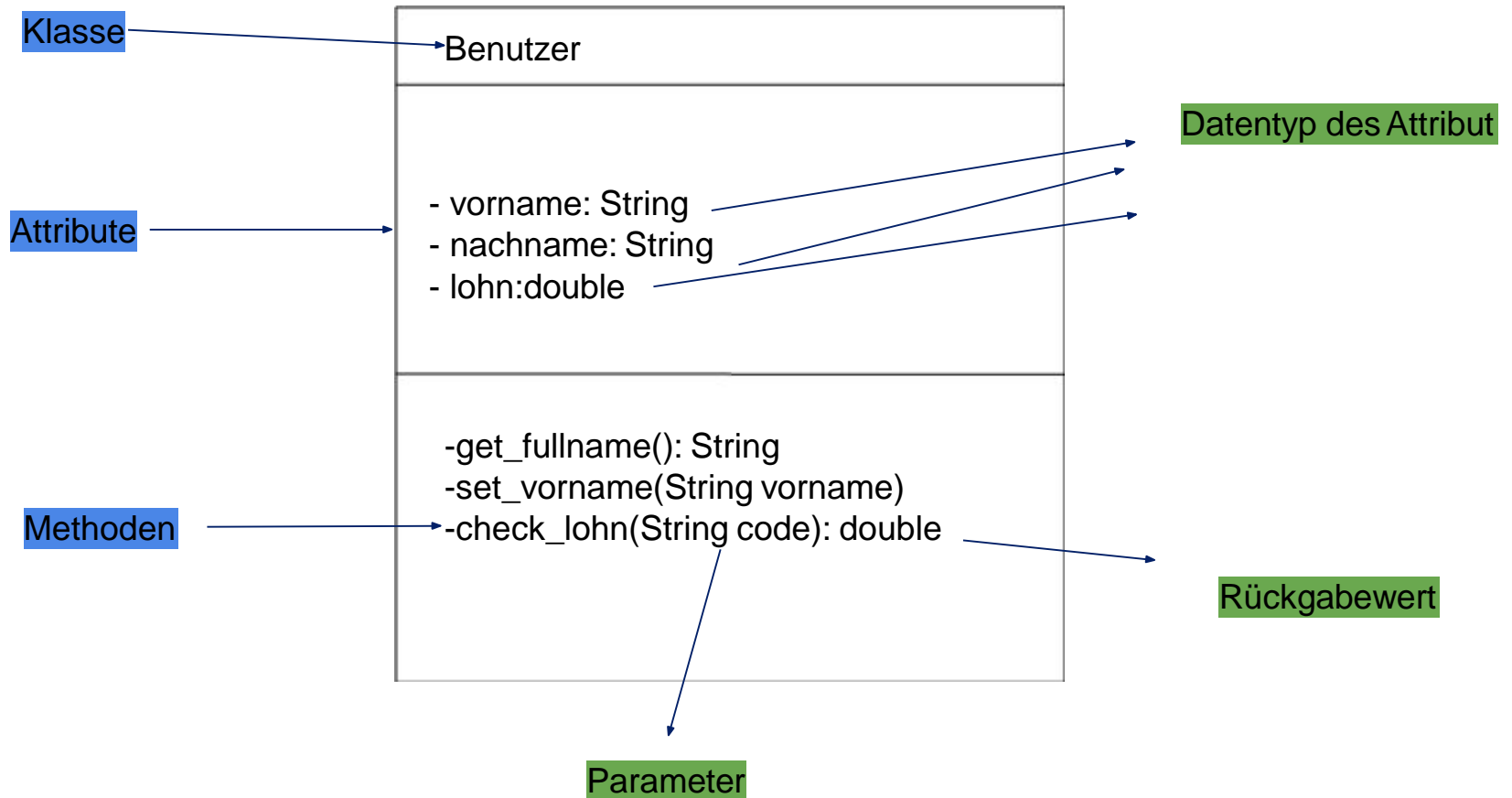
☐ Vervollständigen Sie folgenden Satz:

"Ein Objekt ist eine einer Klasse.

Die Erzeugung eines neuen Objekts nennt man“

☐ Wie nennt man die Werte, die man einer Methode als zusätzliche Angaben übergibt?

Darstellung einer Klasse: Benutzer



Übung Klasse: Bankkonto

Überlegen Sie sich, welches die Methoden und Attribute und eines Bankkontos sind

Referenzen Modell



Was	Einführung in Java Referenzen Modell
Lernziel	Verstehen, wie Python Objekte speichert und adressiert
Warum	Objekte werden dynamisch gespeichert und über Referenzen adressiert
Ablauf	Präsentation Dozent Studenten: Testen Speicheradressen

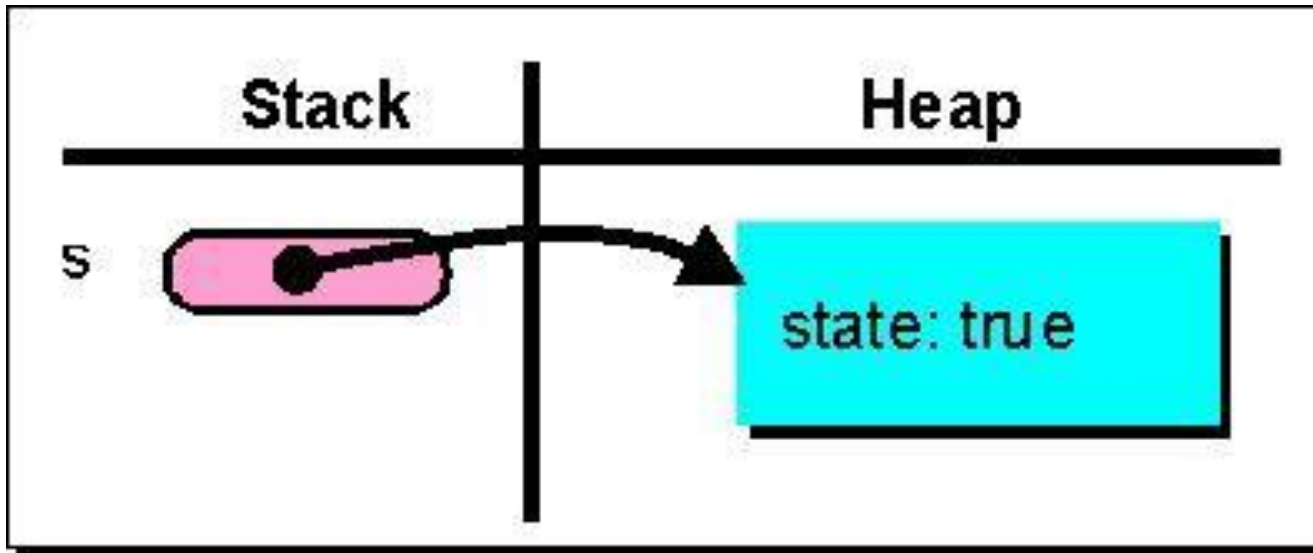
Referenzen Modell - Grundlagen

- ❑ Objekte werden in Python auf dem Heap abgelegt
- ❑ Zugriff auf Objekt erfolgt über Referenzdatentyp
- ❑ Referenz enthält nur Speicheradresse
- ❑ Referenz arbeitet analog dem Zeigerkonzept (Pointer) in C++
- ❑ Jede Referenz ist eindeutig zum Zeitpunkt x
- ❑ Zwei Referenzen zeigen auf das gleiche Objekt, wenn sie zum Zeitpunkt x die gleiche Speicheradresse enthalten
- ❑ Referenzen definieren die Identität der Objekte

Referenzen Modell - Beispiel 1

```
s = Switch()  
print(s)
```

```
# <__main__.Switch object at 0x0171E898>
```



Referenzen Modell - Beispiel 2

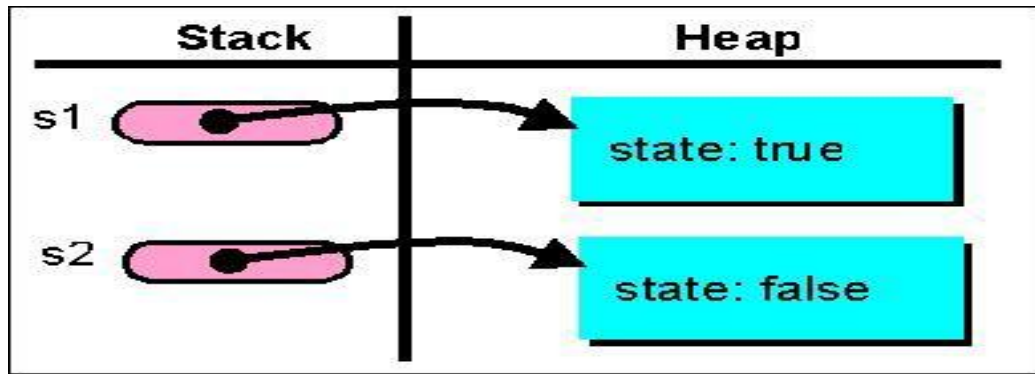
```
s1 = Switch()  
s1.on()  
print(s1)
```

```
s2 = Switch()  
s2.off()  
print(s2)
```

```
# < main .Switch object at 0x0127E868>
```

```
# <__main__.Switch object at 0x012E9358>
```

Referenzen



Referenzen Modell - Frage

```
class Switch:
    def __init__(self):
        self.state = False

    def on(self):
        self.state = True

    def off(self):
        self.state = False

    def is_on(self):
        return self.state
```

```
s1 = Switch()
print(s1)
```

```
s2 = Switch()
print(s2)
```

```
s3 = s1
print(s3)
```

```
s1.on()
```

```
print(s3.is_on()) ---> Was wird die Ausgabe sein ?
```

Klassen in Python



Was	Klassen, Methoden, Attribute kennen lernen
Lernziel	Klassen in Python definieren und nutzen
Warum	OOP Programmierung in Python anwenden können
Ablauf	Präsentation Dozent Studenten: Klassen definieren (Attribute, Methoden)

Klassen - Deklaration einer Klasse

```
class Person:  
    pass
```

- ❑ Nach dem Schlüsselwort `class` steht der Klassenname `Person`
- ❑ `pass` – wird nur benutzt, falls es keinen Inhalt gibt - gilt auch für Methoden die leer sind

Klassen - Konstruktor (Initialisierung)

```
class Person:  
    def __init__(self, vorname):  
        self.vorname = vorname
```

- ❑ `__init__` zur Initialisierung einer Klasse (wird fälschlich auch Konstruktor genannt). Spezielle Funktion
- ❑ `self` – Referenz auf das Objekt (sich selber)
- ❑ `vorname` - Parameter
- ❑ `self.vorname` - Attribut der Klasse Person

Klassen - Erzeugung von Objekten

```
class Person:
    def __init__(self, vorname):
        self.vorname = vorname

    def vorstellen(self):
        print("Hallo, ich heiße", self.vorname)

p = Person("Max")
p.vorstellen()
```

- ❑ `p = Person("Max")`
Ein Objekt `p` wird mit einer Klasse `Person("Max")` angelegt
- ❑ `"Max"` - ist der Parameter, der als Attribut gespeichert werden soll
- ❑ `p.vorstellen()` Die Methode `vorstellen()` des Objektes `p` aufrufen.
- ❑ Nutzung erfolgt
 - klassenname.attribut
 - klassenname.methode

Übung Klasse Rechteck 1:

- ☐ lege eine Klasse 'Rechteck' an
- ☐ beim Instanziiieren eines Rechtecks sollen 'hoehe' und 'breite' als Parameter mitgegeben werden
- ☐ lege eine Instanz Rechteck an
- ☐ Gib die Höhe und die Breite des Rechtecks aus...

Klassen - Methoden

```
class Person:
    def __init__(self, vorname):
        self.vorname = vorname

    def vorstellen(self):
        print("Hallo, ich heiße",
self.vorname)
```

- ❑ `def vorstellen(self)` - Methode (Parameter `self`) muss immer mitgegeben werden (bei Instanzmethoden)
- ❑ `self.vorname` - Attribut der Klasse Person, wird nun benutzt
- ❑ Und jetzt laufen lassen ?

Übung Klasse Rechteck 2:

- ☐ lege eine Klasse 'Rechteck' an
- ☐ beim Instanzieren eines Rechtecks sollen 'hoehe' und 'breite' als Parameter mitgegeben werden
- ☐ lege 3 Methoden an, set_hoehe(), set_breite() und get_Flaeche(). Die Set-Methoden setzen neue Werte
- ☐ die get_Flaeche() solle die Fläche des Rechtecks berechnen und zurückgeben
- ☐ lege eine Instanz Rechteck an
- ☐ Gib die Höhe und die Breite des Rechtecks aus...
- ☐ Verändere beliebig das Rechteck (Höhe oder Breite)
kontrolliere die Fläche jeweils

OOP - Information Hiding



Was	OOP - Information Hiding (Kapselung)
Lernziel	Attribute / Daten bestimmen und kontrolliert freigeben
Warum	Daten kapseln - Zugriff regeln, Kontrolle behalten
Ablauf	Präsentation Dozent Studenten Übungen für Kapselung

Kapselung - Zugriffe regeln

❑ **public**

öffentliche Attribute - keine Einschränkung bei der Nutzung
im Code → `self.inhaber` (kein Unterstrich nach dem Punkt)

❑ **protected**

Attribute nur für die eigene Klasse oder von abgeleiteten Klassen nutzbar. Im Code `self._kontostand` (1 Unterstrich nach dem Punkt)

❑ **private**

Attribute können nur in der eigenen Klasse genutzt werden
Im Code `self.__pin` (2 Unterstriche nach dem Punkt)

Übung

Sichtschutz der Attribute bestimmen

```
class Konto(object):  
    def __init__(self, kontostand):  
        self._kontostand = kontostand  
        self.inhaber = "Felix Muster"  
        self.__pin = "-adsfadfj6663"  
  
    def einzahlen(self, betrag):  
        self._kontostand += betrag  
  
    def auszahlen(self, betrag):  
        self._kontostand -= betrag  
  
k = Konto(22)  
print(k.inhaber)  
print(k._kontostand)  
print(k.__pin)
```

Welche public Attribute ?

Welche protected Attribute ?

Welche private Attribute ?

Bestimmen sie die
Sichtbarkeit der Attribute.

Probieren sie dieses
Programm aus.

Bei k.__pin → gibt es einen
Fehler - warum ?

Übung Klasse Mitarbeiter 1:

- ☐ lege eine Klasse 'Mitarbeiter' an
- ☐ beim Instanzieren eines Mitarbeiters soll 'name' und 'lohn' als Parameter mitgegeben werden
- ☐ der 'name' ist ein öffentliches Attribut, der 'lohn' ist ein privates Attribut
- ☐ lege eine Instanz Mitarbeiter an
- ☐ gib den Namen und den Lohn aus ? Was stellen sie fest ?

Zugriff mit set - und get - Methoden

- ❑ In anderen objektorientierten Programmiersprachen werden Attribute nur mittels Methoden zur Verfügung gestellt
- ❑ get - und set - Methoden werden Attribute gelesen, bzw. neue Werte zugewiesen
- ❑ Python erlaubt/fördert den Zugriff direkt auf Attribute, ohne get/set - Methoden. Aber es soll nicht verschwiegen werden, dass das auch in Python viele Programmierer get/set-Methoden realisieren.
- ❑ Der typische Python Weg, bei protected und private Attribute lautet: Zugriff via Property-Zugriff (wird “Kapitel Property” näher behandelt)

Übung Klasse Mitarbeiter 2:

- ☐ lege eine Klasse 'Mitarbeiter' an
- ☐ beim Instanzieren eines Mitarbeiters soll 'name' und 'lohn' als Parameter mitgegeben werden
- ☐ der 'name' ist ein öffentliches Attribut, der 'lohn' ist ein privates Attribut
- ☐ lege eine Instanz Mitarbeiter an
- ☐ Gib den Namen und den Lohn mittels einer Methode get_lohn()
Was stellen sie jetzt fest ?

OOP - Vererbung



Was OOP - Vererbung

Lernziel Vererbung von Klassen definieren und nutzen
Interaktion von Klassen verstehen

Warum Die Interaktion zwischen Objekten in Python verstehen

Ablauf Präsentation Dozent
Studenten Übungen für Statische Methoden

OOP - Vererbung

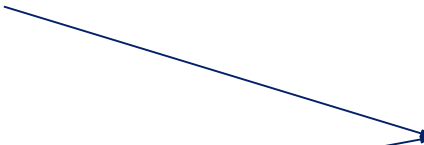
- ❑ Klassisches Konzept der Objektorientierten Programmierung
- ❑ Basisklasse (Superklasse, Elternklasse, Oberklasse) hat eine oder mehrere Subklassen
- ❑ Basisklasse hat eine allgemeine Beschreibung (Attribute / Methoden)
- ❑ Alle Attribute / Methoden werden an die Subklassen vererbt
(wobei der Sichtschutz mögliche Zugriffe beschränkt)
- ❑ Subklasse ist ein Spezialfall der Basisklasse
- ❑ Hohe Wiederverwendbarkeit

OOP Basisklasse

- ❑ Jede Klasse erbt von der, in Python definierten Klasse `object`
(Klassennamen beginnen immer mit Grossbuchstaben, leider ist dies gerade hier nicht so !)
- ❑ Die Klasse `object` beschreibt allgemeinen den Lebenszyklus mit den Methoden `__new__()`, `__init__()` und `__del__()`
- ❑ Jede abgeleitete Klasse kann diese Methoden überschreiben
(in Python wird üblicherweise nur `__init__()` und `__del__()` überschrieben)

```
class Konto:  
    pass
```

```
class Konto(object):  
    pass
```



Gleichwertig,
1. Beispiel: implizit
2. Beispiel: explizit

OOP Vererbung (nur Klasse)

Basisklasse (leer)

```
class Konto:  
    pass
```

- ❑ Basisklasse definiert

Subklasse die von Konto erbt (leer)

```
class Sparkonto(Konto):  
    pass
```

- ❑ Subklasse Sparkonto wird abgeleitet in Klammern die Basisklasse(n)

Instanziierung eines Sparkonto

Kontrolle durch den print

*(<__main__.Sparkonto object at
0x01861880>)*

```
spar = Sparkonto()
```

```
print(spar)
```

- ❑ Es wird ein Sparkonto instanziiert
- ❑ Beweis: print - Referenz

OOP Vererbung (Attribute)

```
## Basisklasse mit 1 Instanzvariable  
(public Attribute)
```

```
class Konto:  
    def __init__(self, inhaber):  
        self.inhaber = inhaber
```

```
# Subklasse die von Konto erbt
```

```
class Sparkonto(Konto):  
    pass
```

```
# Instanziierung eines Sparkonto
```

```
# Kontrolle durch den print - Aufruf  
("Max")
```

```
spar = Sparkonto("Max")  
print(spar.inhaber)
```

❑ Basisklasse definiert, mit 1 Attribut

❑ Subklasse Sparkonto wird abgeleitet in Klammern die Basisklasse(n)

❑ Es wird ein Sparkonto instanziiert

❑ das Attribut `inhaber` ist nutzbar

Übung Klasse Kreditkarte 1:

- ☐ erstellen sie folgende Klassen 'Kreditkarte', 'Visa' und 'Mastercard'
- ☐ beim Instanziiieren soll die Kartennummer mitgegeben werden, die Kartennummer ist public
- ☐ die Klassen 'Visa' und 'Mastercard' erben von 'Kreditkarte' und haben keine eigene Implementation (pass)
- ☐ Instanziiieren sie jeweils eine Visa- und ein Mastercard
- ☐ Geben sie jeweils die Kartennummer aus

OOP Vererbung (Methoden)

```
class Konto:
    def __init__(self, inhaber):
        self.inhaber = inhaber

    def say_hello(self):
        print("Hallo, ich bin", self.inhaber)

# Subklasse die von Konto erbt
class Sparkonto(Konto):
    pass

# Instanziierung eines Sparkonto
# Kontrolle durch den say_hello() Aufruf
spar = Sparkonto("Max")
spar.say_hello()
```

- ❑ Basisklasse definiert, mit 1 Attribut und 1 Methode
- ❑ Subklasse Sparkonto wird abgeleitet in Klammern die Basisklasse(n)
- ❑ Es wird ein Sparkonto instanziiert die Methode `say_hello()` wird

Übung Klasse Kreditkarte 2:

- ☐ erstellen sie folgende Klassen 'Kreditkarte', 'Visa' und 'Mastercard'
- ☐ beim Instanziieren soll die Kartennummer mitgegeben werden, die Kartennummer ist public
- ☐ die Klassen 'Visa' und 'Mastercard' erben von 'Kreditkarte' und haben jeweils einen eigenen Initialisierung
- ☐ bei der Initialisierung der Visa Kartennummer wird die Endung "-1944" angefügt, bei der Initialisierung der Mastercard Kartennummer wird die Endung "-1234" angefügt
- ☐ Instanziieren sie jeweils eine Visa- und ein Mastercard und eine Kreditkarte !
- ☐ Geben sie jeweils die Kartennummer aus

Übung Klasse Kreditkarte 3:

- ☐ erstellen sie folgende Klassen 'Kreditkarte', 'Visa' und 'Mastercard'
- ☐ beim Instanzieren soll die Kartennummer mitgegeben werden, die Kartennummer ist public
- ☐ die Klassen 'Visa' und 'Mastercard' erben von 'Kreditkarte' und haben jeweils einen eigenen Initialisierung
- ☐ bei der Initialisierung der Visa Kartennummer wird die Endung "-1944" angefügt, bei der Initialisierung der Mastercard Kartennummer wird die Endung "-1234" angefügt
- ☐ Definieren sie innerhalb von Kreditkarte eine Methode validate() ohne Parameter - sie retourniert immer "Karte ok"
- ☐ Instanzieren sie jeweils eine Visa- und ein Mastercard und eine Kreditkarte !
- ☐ Validieren sie jede Kreditkarte

OOP Methoden überschreiben

```
class Konto:
    def __init__(self, inhaber):
        self.inhaber = inhaber

    def say_hello(self):
        print("Hallo, ich bin", self.inhaber)

# Subklasse die von Konto erbt
class Sparkonto(Konto):

    def say_hello(self):
        print("Hello, i am", self.inhaber)

# Instanziierung eines Sparkonto
# Kontrolle durch den print - Aufruf ("Max")
spar = Sparkonto("Max")
spar.say_hello()
```

- ❑ Basisklasse hat die Methode `say_hello()`
- ❑ Subklasse hat die Methode `say_hello()`. Somit wurde die Methode der Basisklasse überschrieben
- ❑ Es wird ein Sparkonto instanziiert die Methode `say_hello()` wird ausgeführt der Klasse Sparkonto

Übung Klasse Kreditkarte 4:

- ☐ erstellen sie folgende Klassen 'Kreditkarte', 'Visa' und 'Mastercard'
- ☐ beim Instanzieren soll die Kartennummer mitgegeben werden, die Kartennummer ist public
- ☐ die Klassen 'Visa' und 'Mastercard' erben von 'Kreditkarte' und haben jeweils einen eigenen Initialisierung
- ☐ bei der Initialisierung der Visa Kartennummer wird die Endung "-1944" angefügt, bei der Initialisierung der Mastercard Kartennummer wird die Endung "-1234" angefügt
- ☐ definieren sie innerhalb von Kreditkarte eine Methode validate() ohne Parameter - sie retourniert immer "Karte ok"
- ☐ definieren sie innerhalb von Visa bzw. Mastercard jeweils Methode validate() ohne Parameter - führen sie darin einen beliebigen Check durch und geben jeweils "Karte ok" oder "Karte no" zurück
- ☐ Instanzieren sie jeweils eine Visa- und ein Mastercard und eine Kreditkarte !
- ☐ validieren sie jede Kreditkarte

OOP Klassenattribute


```
class Konto:
    zinssatz = 0.5

    def __init__(self, inhaber):
        self.inhaber = inhaber

    def say_hello(self):
        print("Hallo, ich bin", self.inhaber)

print(Konto.zinssatz)
k = Konto("Max")
print(k.zinssatz)

Konto.zinssatz = 0.75
print(k.zinssatz)
```



- ❑ Zinssatz ist eine Klassenvariable
Sie ist fix an die Klasse gebunden.
Sie existiert unabhängig von Instanzen. Verhindert Redundanz
- ❑ Klassenattribute immer über **Klassennamen.Attribute** ansprechen !!
- ❑ Wird Klassenattribut über eine Instanz neu initialisiert → dann entsteht ein **dynamisches Attribut** (nächstes Kapitel)

Übung Klassenattribut Kreis:

- ☐ erstellen sie folgende Klasse 'Kreis'
- ☐ beim Instanziiieren soll der Radius mitgegeben werden, der Radius ist public
- ☐ definieren sie ein Klassenvariable $\pi = 3.1$
- ☐ berechnen sie die Fläche in einer Methode und geben diesen Wert zurück
- ☐ geben sie beim Testen den Wert von π aus...
- ☐ instanziiieren sie nun 2 Kreise mit unterschiedlichen Werten
- ☐ drucken von beiden die Fläche aus
- ☐ ändern nun den Wert von π auf $= 3.1415$
- ☐ drucken von beiden die Fläche erneut aus aus

OOP Klassenmethode Methoden

```
class Konto:
    zinssatz = 0.5

    def __init__(self, betrag):
        self.saldo = betrag

    @classmethod
    def change_zins(cls):
        print("Der neue Zins beträgt:",
              cls.zinssatz * 0.99)

print(Konto.zinssatz)
k = Konto(100)
Konto.change_zins()
```

- ❑ `@classmethod`
`def change_zins(cls):` ist eine Klassenmethode
- ❑ Die Annotation `@classmethod` macht aus einer Instanzmethode eine Klassenmethode
- ❑ `cls` Ist der "Klassenname" als Parameter. Über `cls` werden Klassen Ressourcen angesprochen. Kein Zugriff auf `self` - Ressourcen
- ❑ Verhindert Redundanz

Übung Klassenmethode Person:

- ❑ Kopieren sie die Klasse `Person` von der Aufgabe `Static_Aufgabe1`
- ❑ ergänzen sie Klasse `Person` um die Klassenmethode `person_jahrgang` mit `name` und `jahrgang` als Parameter
- ❑ diese Methode berechnet das Alter und gibt eine Instanz `Person` zurück
- ❑ Instanziiieren sie eine `Person` indem sie Methode `person_jahrgang` nutzen
- ❑ Geben sie in einem `print` an ob diese `Person` nun volljährig ist oder nicht

Zugriffe via property regeln (1)

❑ Zugriffe auf private oder protected Attribute

Prüfenswerte Attribute jeweils nur mit property Zugriff regeln.

- ❑ `propname=property(getter, setter, deleter, docstring)`
propname ist frei wählbar. Die Standard Built-in Funktion `property()` bindet die Methoden an die Property ("Propname"). Jeder Zugriff erfolgt, dann via gebundener Methode. Im Programm wird der "Propname" angesprochen. 1 Methode muss gebunden sein

```
def set_name(self, name):  
    self.__name = name
```

```
def get_name(self):  
    return self.__name
```

```
name = property(get_name, set_name)
```

Zugriffe via property regeln (2)

❑ Zugriffe auf private oder protected Attribute

Prüfenswerte Attribute jeweils nur mit @property Zugriff regeln. Die Annotation ist die moderne Variante

```
class Person:
```

```
    def __init__(self, fullname):
```

```
        self.__name = fullname
```

```
@property
```

```
def name(self):
```

```
    return self.__name
```

```
@name.setter
```

```
def name(self, value):
```

```
    self.__name = value
```

getter()



setter()



Property - Zugriffe (1)

```
class Konto(object):  
    def __init__(self, kontostand):  
        self._kontostand = kontostand  
        self.inhaber = "Felix Muster"  
        self.__pin = "-adsfadj6663"  
  
    def einzahlen(self, betrag):  
        self._kontostand += betrag  
  
    def auszahlen(self, betrag):  
        self._kontostand -= betrag  
  
    # neuere Art einer Property  
    @property  
    def kontostand(self):  
        return self._kontostand  
  
    # ältere Art einer Property  
    def pin(self):  
        return "Zugriff verboten"  
  
    # ältere Art einer Property  
    pin = property(pin)
```

❑ `self._kontostand` das protected Attribut wird via einer Property `@property` zur Verfügung gestellt

❑ `self.__pin` das private Attribut wird via einer Property via Methode `pin()` und `property(pin)` zur Verfügung gestellt

❑ Beide Arten sind korrekt und gleichwertig, einfacher und neuer ist mit dem `@property` eine Property zu erstellen

Property - Zugriffe (2)

```
class Konto(object):
    def __init__(self, kontostand):
        self._kontostand = kontostand
        self.inhaber = "Felix Muster"
        self.__pin = "-adsfadj6663"

    # neuere Art einer Property
    @property
    def kontostand(self):
        return self._kontostand

    # ältere Art einer Porperty
    def pin(self):
        return "Zugriff verboten"

    # ältere Art einer Porperty
    pin = property(pin)
```

```
k = Konto(20)
print(k.inhaber)
print(k.kontostand)
print(k.pin)
```

- ❑ `self.inhaber` - normaler Zugriff via Attributname `k.inhaber`
- ❑ `self._kontostand` wird via `k.kontostand` verfügbar
- ❑ `@property` regelt den Zugriff
- ❑ `self.__pin` wird via `k.pin` verfügbar, aber der Inhalt wird über das `property()` "verschleiert"

Übung Temperatur Konverter 1 - “alt” Property

- ❑ erstellen sie eine Klasse 'Celsius'
- ❑ beim Instanziiieren soll die Temperatur in Celsius mitgegeben werden
- ❑ Die Temperatur ist protected
- ❑ die Klasse kann die Temperatur in Fahrenheit ausgeben. Formel $x * 1.8 + 32$
- ❑ die Zugriffe auf Temperatur via getter() und setter() Methode, die mit Property (old-style) gebunden wird
- ❑ die Setter - Methode validiert, ob der Wert als 273 Grad Celsius ist, wenn tiefer --> Error werfen
- ❑ testen sie diese Klasse mit diversen Ausgaben (Celsius und Fahrenheit), ändern sie Temperatur

Übung Temperatur Konverter 2 - @property

- ❑ erstellen sie eine Klasse 'Celsius'
- ❑ beim Instanziiieren soll die Temperatur in Celsius mitgegeben werden
- ❑ Die Temperatur ist protected
- ❑ die Klasse kann die Temperatur in Fahrenheit ausgeben. Formel $x * 1.8 + 32$
- ❑ die Zugriffe auf Temperatur via getter() und setter() Methode, die mit Annotation @property gebunden wird
- ❑ die Setter - Methode validiert, ob der Wert als 273 Grad Celsius ist, wenn tiefer --> Error werfen
- ❑ testen sie diese Klasse mit diversen Ausgaben (Celsius und Fahrenheit), ändern sie Temperatur

OOP - Dynamische Attribute



Was	OOP - Dynamische Attribute
Lernziel	Dynamische Attribute einem Objekt hinzufügen und nutzen
Warum	Objekt zur Laufzeit um Attribute ergänzen zu können
Ablauf	Präsentation Dozent Studenten Übungen für dynamische Attribute

Dynamische Attribute - Beispiel

```
class Konto:
    def __init__(self, inhaber):
        self.inhaber = inhaber

k = Konto("Max")
print(k.inhaber)

# dynamische Attribut an die Instanz gebunden
k.adresse = "8000 Zürich"

print(k.adresse)

m = Konto("Fritz")
print(m.inhaber)
print(m.adresse)  # das Konto m hat keine Adresse !
```

- ❑ der Instanz `k` wird das dynamische Attribut `adresse` angehängt (Die Klasse kennt das Attribut `adresse` nicht)
`k.adresse = "8000 Zürich"`
- ❑ das Attribut wird nur einer Instanz angehängt, andere Instanzen "kennen" dieses Attribut nicht
- ❑ wird benutzt um "einmalige" Ergänzung zu ermöglichen. Gutes Werkzeug, wenn eine bestehende Klasse dynamisch ergänzt werden muss

Übung: Dynamische Attribute

- ☐ erstellen sie eine Klasse Buch, beim Instanziieren geben sie titel (public) und preis (public) mit
- ☐ legen sie eine Instanz Buch an
- ☐ fügen sie ein dynamisches Attribut 'bemerkung' an die die Instanz an
- ☐ geben sie alle 3 Werte aus (titel, preis) und bemerkung
- ☐ für den Test: legen sie eine weiter Instanz an und versuchen sie die Bemerkung auszugeben
- ☐ Wiederholen sie die Schritte 4-6 nochmals mit dem Unterschied, dass sie die Bemerkung der Klasse Buch anfügen
- ☐ Was fällt auf ?

OOP - Dynamische Methoden



Was	OOP - Dynamische Methoden
Lernziel	Dynamische Methoden einer Instanz oder der Klasse dynamische hinzufügen und nutzen
Warum	Objekt zur Laufzeit um Methoden ergänzen zu können
Ablauf	Präsentation Dozent Studenten Übungen für dynamische Methoden

Dynamische Methoden - Beispiel

```
class Konto:  
    def __init__(self, inhaber):  
        self.inhaber = inhaber
```

```
# freie Methode  
def freie_methode(self):  
    return "Freie Methode"
```

```
k = Konto("Max")  
print(k.inhaber)
```

```
# dynamische Methode an die Klasse (Konto) oder k  
an die Instanz gebunden  
Konto.freie_methode = freie_methode  
print(k.freie_methode())
```

```
m = Konto("Fritz")  
print(m.inhaber)  
print(m.freie_methode())
```

❑ `def freie_methode(self):` ist eine freie Funktion

❑ die Funktion wird nun an Konto (Klasse) oder an k (Instanz) gebunden

❑ je nach Bindung gilt sie nur für eine bestimmte Instanz oder für die ganze Klasse. Endet das Programm endet die Bindung (dynamische Bindung)

Übung: Dynamische Methode

- ☐ Erstellen sie eine Klasse Buch, beim Instanziieren geben sie titel (public) und preis (public) mit
- ☐ Legen sie eine Instanz Buch an
- ☐ Fügen sie eine dynamische Methode 'rabatt' die nur die Zahl 50 zurückgibt an die Instanz an
- ☐ Geben sie alle 3 Werte aus (titel, preis) und rabatt
- ☐ Für den Test: legen sie eine weiter Instanz an und versuchen sie den Rabatt auszugeben
- ☐ Wiederholen sie die Schritte 4-6 nochmals mit dem Unterschied, dass sie die Rabatt der Klasse Buch anfügen
- ☐ Was fällt auf ?

OOP - Statische Methoden



Was OOP - Statische Methoden

Lernziel Statische Methoden definieren und nutzen

Warum Redundanz vermeiden

Ablauf Präsentation Dozent
Studenten Übungen für Statische Methoden

Statische Methoden - Beispiel

```
class Taschenrechner:
```

```
    @staticmethod
```

```
    def add(a, b):
```

```
        return a + b
```

```
    @staticmethod
```

```
    def minus(a, b):
```

```
        return a - b
```

```
    @staticmethod
```

```
    def divide(a, b):
```

```
        return a / b
```

```
    @staticmethod
```

```
    def multiply(a, b):
```

```
        return a * b
```

```
print(Taschenrechner.add(3, 4))
```

```
print(Taschenrechner.minus(3, 4))
```

```
print(Taschenrechner.divide(3, 4))
```

```
print(Taschenrechner.multiply(3, 4))
```

- ❑ `@staticmethod` - sind statische Methoden, die weder auf Ressourcen der Instanz, noch der Klasse zugreifen können
- ❑ Statische Methoden immer über die **Klasse.Methode** ansprechen
- ❑ Wo statische Methoden ?
Thematisch gehören diese Methoden zur Klasse ! Meistens verwendet für nützliche "Hilfsfunktionen"

Übung Statische Methoden:

- ❑ erstellen sie die Klasse `Person`
- ❑ beim Instanziiieren soll der `Name` und das `Alter` mitgegeben werden, der `Name` und `Alter` sind `public`
- ❑ definieren sie statische Methode `volljährig` mit Parameter `alter`. Diese Methode prüft ob jemand älter 18 ist → return `True`, ansonsten `False`.
- ❑ rufen sie diese Methode `volljährig` auf und drucken das Ergebnis auf die Konsole

Abschluss - OOP mit Python

- ❑ Leider konnte nicht alles behandelt werden
- ❑ Begriffe und Anwendung von OO braucht viel Übung
- ❑ Viele Details wurden aus zeitlichen und didaktischen Gründen, nicht erläutert
→ dennoch sind sie auf hohem Niveau
- ❑ Wenn sie einen Hammer haben, sehen sie ev. überall Nägel ? Machen sie sich Gedanken ob OO mittels Python die Lösung ist oder wird es nur kompliziert
- ❑ OO ist ein Instrument / Konzept - und kann viel Spass machen
- ❑ Viel Erfolg mit Python !